Ai_Curl_Tracker

June 12, 2022

1 Install and Import Dependencies

[ ]:
```
pip install mediapipe opencv-python
#mediapipe
```

[2]:
```
import cv2
import mediapipe as mp
import numpy as np
#numpy is a python library that provide N-D Array
mp_drawing=mp.solutions.drawing_utils
# mp.solutions.drawing_utils class  will allow us to visualize the landmarks␣
 ↪after detection,
mp_pose=mp.solutions.pose
#Include pose library
```

[10]:
```
#Video Feed
cap=cv2.VideoCapture(0)
#VideoCapture() method of cv2 library is used  to read and start live streaming.
 ↪ Its Possible values if either 0 and -1
#device index ie 0 : It is just the number to specify the camera.
while cap.isOpened:
    ret,frame=cap.read()
    #When we apply command cap.read() the first frame from our video file will␣
 ↪be loaded.
    #It will be stored in a variable frame. If we call this command again, the␣
 ↪second frame will be loaded and so on.
    #Variable ret is a boolean data type that returns True if we are able to␣
 ↪execute the read function successfully.
    #ret will obtain return value getting from the camera frame either true or␣
 ↪false
    cv2.imshow('Mediapipe Feed',frame)
    #Displays an image in the specified window.

    if(cv2.waitKey(10) & 0xFF==ord('q')):
    #delay
    # Waits for a pressed key.
    #where 10 is the delay in miniseconds
```

```
    #if we press 'q' then it will return in  string but we need answer in␣
 ↪binary form
    # so we use hexadecimal 0xFF i.e 255 in decimal, so it will convert string␣
 ↪into binary
    #with the help of and operation
      break

cap.release()
# Closes video file or capturing device.
cv2.destroyAllWindows()
#Destroys all of the HighGUI windows.
```

2 Make Detections

[14]:
```
cap=cv2.VideoCapture(0)
#setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5 ,min_tracking_confidence=0.5) as␣
 ↪pose:
    # It is used to specify the minimum confidence value with which the␣
 ↪detection from the landmark-tracking model
    #must be considered as successful.
    #Its default value is 0.5
    #Setting it to a higher value can increase robustness of the solution, at␣
 ↪the expense of a higher latency.
    while cap.isOpened():
        #Returns true if video capturing has been initialized already.
        ret,frame=cap.read()
        #Grabs, decodes and returns the next video frame.
        #ret will obtain return value getting from the camera frame either true␣
 ↪or false


        #Recolor image to RGB
        image=cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
        #cvtColor:- Converts an image from one color space to another.
        image.flags.writeable=False


        #Make Detections
        results=pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

```
        #Render Detections
        # Draw landmarks
        mp_drawing.draw_landmarks(image,results.pose_landmarks,mp_pose.
↪POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66),␣
↪thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230),␣
↪thickness=2, circle_radius=2)
                                   )
        #Drawingspec:-Draws the detection bounding box and keypoints on the␣
↪image
        cv2.imshow('Mediapipe Feed', image)
        if cv2.waitKey(10) & 0xFF == ord('q'): # # Break gracefully
            break

    cap.release()
    cv2.destroyAllWindows()
```

[17]:
```
#mp_drawing.DrawingSpec??
```

3 Determining Joints

[ ]:
```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as␣
↪pose:
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True

        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark
            print(landmarks)
        except:
            pass
```

```
        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.
↪POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66),␣
↪thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230),␣
↪thickness=2, circle_radius=2)
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

[21]:
```
len(landmarks)
```

[21]: 33

[23]:
```
for lmark in mp_pose.PoseLandmark:
    print(lmark)
```

```
PoseLandmark.NOSE
PoseLandmark.LEFT_EYE_INNER
PoseLandmark.LEFT_EYE
PoseLandmark.LEFT_EYE_OUTER
PoseLandmark.RIGHT_EYE_INNER
PoseLandmark.RIGHT_EYE
PoseLandmark.RIGHT_EYE_OUTER
PoseLandmark.LEFT_EAR
PoseLandmark.RIGHT_EAR
PoseLandmark.MOUTH_LEFT
PoseLandmark.MOUTH_RIGHT
PoseLandmark.LEFT_SHOULDER
PoseLandmark.RIGHT_SHOULDER
PoseLandmark.LEFT_ELBOW
PoseLandmark.RIGHT_ELBOW
PoseLandmark.LEFT_WRIST
PoseLandmark.RIGHT_WRIST
PoseLandmark.LEFT_PINKY
PoseLandmark.RIGHT_PINKY
PoseLandmark.LEFT_INDEX
PoseLandmark.RIGHT_INDEX
```

```
PoseLandmark.LEFT_THUMB
PoseLandmark.RIGHT_THUMB
PoseLandmark.LEFT_HIP
PoseLandmark.RIGHT_HIP
PoseLandmark.LEFT_KNEE
PoseLandmark.RIGHT_KNEE
PoseLandmark.LEFT_ANKLE
PoseLandmark.RIGHT_ANKLE
PoseLandmark.LEFT_HEEL
PoseLandmark.RIGHT_HEEL
PoseLandmark.LEFT_FOOT_INDEX
PoseLandmark.RIGHT_FOOT_INDEX
```

[24]:
```
landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].visibility
```

[24]:
```
0.9353919625282288
```

[26]:
```
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
```

[26]:
```
x: 1.0102804899215698
y: 1.2574375867843628
z: -1.1176338195800781
visibility: 0.17017368972301483
```

[27]:
```
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
```

[27]:
```
x: 0.9589055180549622
y: 1.3104153871536255
z: -1.8593040704727173
visibility: 0.030339263379573822
```

4 ANGLES CALCULATING

[28]:
```
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1],␣
 ↪a[0]-b[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle
```

[29]:
```
shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].
 ↪x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.
 ↪PoseLandmark.LEFT_ELBOW.value].y]
wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.
 ↪PoseLandmark.LEFT_WRIST.value].y]
```

[31]:
```
shoulder, elbow, wrist
```

[31]:
```
([0.8768285512924194, 0.9720253348350525],
 [1.0102804899215698, 1.2574375867843628],
 [0.9589055180549622, 1.3104153871536255])
```

[33]:
```
calculate_angle(shoulder, elbow, wrist)
```

[33]:
```
110.82031204327163
```

[34]:
```
tuple(np.multiply(elbow, [640, 480]).astype(int))
```

[34]:
```
(646, 603)
```

[35]:
```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as␣
 ↪pose:
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].
 ↪x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
```

```
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            # Calculate angle
            angle = calculate_angle(shoulder, elbow, wrist)
            # Visualize angle
            cv2.putText(image, str(angle),
                           tuple(np.multiply(elbow, [640, 480]).astype(int)),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2,␣
↪cv2.LINE_AA
                                )

        except:
            pass


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.
↪POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66),␣
↪thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230),␣
↪thickness=2, circle_radius=2)
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

5 Curl Counter

[39]:
```
cap = cv2.VideoCapture(0)

# Curl counter variables
counter = 0
stage = None

## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as␣
↪pose:
```

```
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].
↪x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            # Calculate angle
            angle = calculate_angle(shoulder, elbow, wrist)

            # Visualize angle
            cv2.putText(image, str(angle),
                           tuple(np.multiply(elbow, [640, 480]).astype(int)),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2,␣
↪cv2.LINE_AA
                                )

            # Curl counter logic
            if angle > 160:
```

```
                stage = "down"
            if angle < 30 and stage =='down':
                stage="up"
                counter +=1
                print(counter)

        except:
            pass
        # Render curl counter
        # Setup status box
        cv2.rectangle(image, (0,0), (225,73), (245,117,16), -1)

        # Rep data
        cv2.putText(image, 'REPS', (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, str(counter),
                    (10,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)

        # Stage data
        cv2.putText(image, 'STAGE', (65,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, stage,
                    (60,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)
        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.
↪POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66),␣
↪thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230),␣
↪thickness=2, circle_radius=2)
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```